Milwaukee Area Atari Users Group

Vol.5 Nbr.3

MILATARI NEWSLETTER

Feburary, 1986

PRICE $1.50

Calender of Events

Feburary 13, 1986 ATR8000 - CP/M SIG

8:00PM - At Joe Kasper's home
Call 782-9041 for adjenda

Feburary 15, 1986 Armbruster School
7000 Greenway, Greendale
(Off 68th, 1 Block North of Grange)

2:15PM - 520ST SIG

2:15PM - NEW BASIC Class - Steve Armstrong

2:15PM - The Learning Phone Workshop
Tom Terwilliger

3:30PM - ELECTRONIC MAIL - Ken Bergren

Feburary 20, 1986 Board of Directors Meeting 7:00PM
Ground Round Bluemound & Hwy 100

March 13, 1986 8PM - ATR8000 - CP/M SIG

ADVENTURER'S ANSWERS

If you don't mind the late night hours and thrive on the challenge, then beware! This column is not for you! If, however, you are ready to give up and can't stand the thought of one more sleepless night, then read on!

MYSTERY FUNHOUSE
from Scott Adams
the solution

A Fun House may be a strange place for espionage, but secret agents have a habit of popping up where you least expect them.

You stand, shoes in hand, outside the Fun House. If you had some money, you could get inside. The first thing to do is drop the watch and go east to the parking lot. Ignore the five dollar bill (it's a grocery bill), and look at the tree.

Get the branch, then look in the grate. Chew the gum (tastes HORRIBLE!), then stick it on the branch. Use the branch to get the coin, then drop both the branch and the gum. Return to the Fun House entrance.

Wear the shoes and give the dollar for a ticket. Go Fun. You are standing in the Magical Mirror Room, and just up ahead is a maze. Go north three times, then West twice. This brings you to a small room. Go west from here to the room with the knobs. Pull the green knob, and you will find yourself in another room. Get the trampoline, then go south to the shooting gallery and pick up the strange spectacles. Go north, then up, to the knob room.

Head along west to the tank room, and from there, up to the ledge over the pit. Drop the trampoline, then go east to the barrel room.

In the barrel room, get the match and the comb, then crawl exit.. Drop the match by the trampoline, then head south to the rickety stairs and down to the landing. Go down the slide into the tank. Get the rusty key, then give the comb to the mermaid. Go up the secret stairs she reveals, and you will be back on the landing.

Go east into the Windy Hall, and east again into the maze. Now, carefully make your way south, east, south, east, into the Mirror Room. Wear the spectacles, and look in the mirror. There's a hidden door! Open the door and go inside. Here you find a valve handle. Drop the spectacles and get the handle, then go East back to the mirror room.

Go through the maze to the room with the knobs. Drop the key, then continue west and up until you come to the ledge. Get the trampoline, then go down the ladder to the pit. Drop the trampoline, then put handle and turn handle. You have now turned off the calliope in the merry-go-round room. Go trampoline, and jump. Wheeee! You're back up on the ledge.

Get the match, and return to the knob room. Drop the match, get the key, and pull the blue knob. Now you're in the room with the Fortune-telling machine. Go east to the merry-go-round room.

Once in the merry-go-round room, push the blue button to stop the ride. Go merry, then go horse. Climb the pole in the horse's back, which brings you to the top of the ride. Look up, and you will see a rope. Jump! Now look, and you will see you are on a catwalk. Go east, and unlock the door. Drop the key, and look at the shelves. Grab the flashlight and the wrench, then head west, and climb all the way down again, then return to the knob room. As you pass the fortune-telling machine, pick up the "out of order" sign.

Pull the green knob, then go south to the shooting gallery. Drop the sign there, then return to the knob room. Get the match, then pull the yellow knob. This takes you to a small room with strange music. Go north into the maze, then go east, south, east, south, east, and you're in the Mirror Room. From there, go south out of the Fun House.

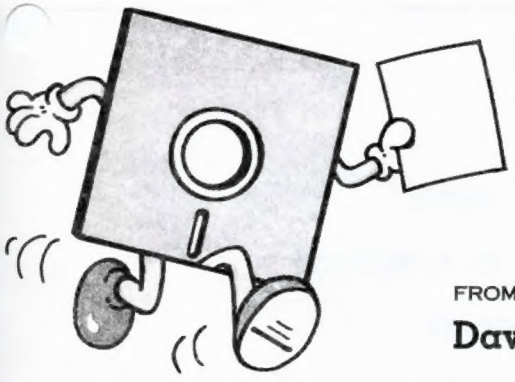
It's back to the parking lot. Open the grate. You will only be able to open one bolt, but you can slide the grate to make room for yourself to go down. Drop the wrench and get the gum. Turn on the flashlight, then go down the manhole, and east to the room with the second grate.

Close the door, and drop the ticket. Remove the heel from your shoe. A couple of things will fall out. One is a note expaining what this is all about, the other is a fuse. Drop the heel and the note, and get the fuse.

Chew the gum again, then stick it to the fuse, and then to the grate. Light the fuse, and POP! the grate blows open. Get the ticket, then go through the hole. Now up through the shooting gallery and south to the hidden lab. The secret plans are there! Grab them.

CONGRATULATIONS!! You have successfully completed your mission!

Reprinted from Computer Squad



FROM THE DISK OF

Dave Frazer

MILATARI LTD. is soon to be born.

We have been in the process of incorporation for several months. The papers are now on their way to Madison for processing.

Milatari, soon to be known as MILATARI LTD., with be a non-profit educational corporation. As soon as we receive our approval from the state, we can proceed with filing for non-profit status with the Internal Revenue Service.

After we have received the IRS approval we can begin acting like a non-profit organization. That is, will be able to mail at a lower postage rate, we can receive tax deductible donations and gifts and the paper work for the treasurer will increase 25 fold.

Watch for our new name - but the game will remain the same!

FEBRUARY MEETING PLANS

Get your rollerskates ready folks, we have many activities planned for our February meeting. The problem is that there are some overlaps - so you many need to bring grandma along to take notes.

At 2:15 PM the following activities will take place;

520ST SIG: Gary Nolan hosts another information sharing session for you owners and soon-to-be owners of the Atari 520ST computer. (Or should we rename to the 1040/520ST SIG?)

LEARNING PHONE WORKSHOP: Tom Terwilliger will present an on-line demonstration of Control Data's renamed PLATO system. The network provides VIDEO-TEX products for education. The available data bases serve children through adult learning needs.

BASIC CLASS: Steve Armstrong leads a new series of basic classes for the membership. The cost is $5.00 per class. Be prepared to stretch your BASIC knowledge.

At 3:30 we are pleased to have Ken Bergren join us. His presentation will on electronic mail services.

Ken is the president of Electronic Mail Solutions, Inc. EMS represents both MCI Mail and Western Union's EasyLink electronic mail service in Wisconsin.

At 4:15 we will hold our regular club business meeting.

(Continued on page 6)

MEM BASIC Classes Taught by Steve Armstrong Consulting Hours
3346 S. 92nd 543-9039 9-12AM Sat. for students

FEB. Session: Exploring Applications and Model design
Feb. 14 2:15PM Armbruster
Cost: $5 + $3 for disk with listings

Future Sessions include: Geometry, Biology/Psychology, Database - Games

*** CHRIS CRAWFORD *** ASSEMBLY LANGUAGE COURSE

LESSON SEVEN: INTERRUPTS

We now approach one of the most difficult topics in the world of assembly: interrupts. This is such a messy topic that very few high-level languages make any provision for interrupts. Moreover, interuppts are one of the best ways around crash your program hard. Programmers using interrupts must be very careful.

The standard way to handle this problem is with a technique called polling. Your program runs out every now and then to check whether the high-priority situation has arisen. If it has, then the program responds to it. If not, it returns to its original work.

The problem with polling arises from the choice of polling interval. If you choose a long (infrequent) polling interval, then you may not respond to a demand quickly enough. If you choose a short (frequent) polling interval, then you will respond quickly to the demand, but you will never have any time for your regular computations.

You may think this type of situation is infrequent, but I can list quite a few situations where this is fairly common. Most I/O operations involve short bursts of computation at infrequent intervals, but they must be attended to on a tight schedule. For example, talking to a cassette deck involves very little real work from the CPU, but it must be done according to a precise schedule.

Even a disk drive is very slow by the standards of a 6502. Or how about keyboard response? When the human operator presses a button, he wants to see response NOW, not two or three seconds from now. Yet he could press that button at any time. So should your program sit on its hands waiting for a keypress, or should it ignore the human operator?

The solution to all of these problems is the interrupt. An interrupt is rather like a subroutine that can be called by a hardware action. There's a wire going into the 6502 called IRQ (Interrupt Request). That wire is normally quiet. But when something important happens, like a keypress, the computer's hardware puts a signal on that wire to interrupt the 6502. Here's what happens next:

The 6502 is busy running a program, but when it gets the interrupt signal it first checks the 1-bit (Interrupt) in the processor status register. If the 1 bit is set, it decides to ignore the interrupt, but if it is clear, it proceeds to the next step. It saves the processor status register and the current value of the program counter onto the stack.

Then it loads the program counter with the address stored at a special place in ROM -- it's either $FFFC or $FFFE, I can never giet it straight. It thus jumps to the address specified in ROM. It expects to find an interrupt service routine there, which

presumably will deal with the keypress in the appropriate manner.

This routine will probably start by pushing A, X. and Y into the stack to preserve them. When done, the routine will then pull them off the stack and execute an RTI instruction, which causes the 6502 to pull the processor status register off the stack, and then pull the program counter off and resume operating.

The important thing about this rather complex sequence is that it allows the 6502 to drop whatever it is doing, service the interrupt, and then return to its earlier functioning without skipping a beat. The overriding goal of all this is to be absolutely certain that, when the 6502 reeturns from the interrupt, it returns in EXACTLY the same state that it was in when the interrupt hit. Otherwise, all sorts of horrible, untraceable bugs would result.

Imagine -- you're in the middle of some huge computation when an interrupt strikes. It subtly changes some very tiny parameter, just enough to insure that when the computation resumes, it will be slightly incorrect. When you try to find the bug, you discover that sometimes the code works perfectly and sometimes it fouls up, and you can't figure out why it should do that. Very bad business!

Moral: interrupts must follow a very tight discipline if they are to be of any utility.

Now let's get into some of the technical gore involving interrupts. First, there are two interrupts on the 6502. They are called IRQ (Interrupt Request) and NMI (Non-Maskable Interrupt). The idea is that the IRQ cterppts are on of the best ways around crash your program hard. Programmers using interrupts must be very careful.

The standard way to handle s problem is with a technique called polling. Your program runs out every now and then to check whether the high-priority situation has arisen. If it has, then the program responds to it. If not, it returns to its original work.

The problem with polling arises from the choice of polling interval. If you choose a long (infrequent) polling interval,puters don't allow that.

The NMI and IRQ interrupts have separate interrupt vectors in ROM, so they can be treated differently. These vectors route the interrupts to the OS, but the OS is smart enough to route interrupt flow thorugh some RAM locations. This means that you can intercept these two interrupts by altering the contents of the RAM-vectors. (I won't list them here, there are a number of them for different situations.)

You must be careful, though, when altering such a RAM vector. What happens if an interrupt strikes after you have changed one byte of the address and before you have changed the other byte? The 6502 will fly off into never-never land and you have crashed. Sure, it's unlikely, but good programmers don't count on luck to make their programs work. You have to guarantee that the interrupt won't occur before you mess with the vector. Use SETVBV from the OS.

The two primary applications of interrupts with the Atari computers are for VBIs (Vertical Blank Interrupts) and DLIs (Display List Interupts). These are very involved topics covered quite thoroughly in th book De Re Atari. VBIs are most often used for animation control, input handling, and other time-critical operations. For example, the entire player I/O of my game Eastern Front (1941) is handled by VBIs. The scrolling, giving of orders, identifying units, and so forth is all done by VBIs. The

mainline routine meanwhile figures the artificial intelligence.

DLIs are used to enhance the graphics on the screen. You can get more colors, more use out of players, more scrolling, and more character sets with proper use of DLIs. Again, consult De Re Atari for a full treatment of this complex subject.

Interrupts are extremely difficult to debug because they tend to crash the system when they fail. You must exercise the strictest discipline in writing interrupt code. Timing problems, seldom of concern in mainline programming, can become critical with interrupts.

What happens, for example, if your interrupt service routine takes so much execution time that more interrupts arrive than you can service? Bad things, I assure you. You must always ask yourself, what happens if an interrupt strikes here? Or there? You must assume that an interrupt will strike at the worst possible time, and write your code to deal with that possibility.

The most important discipline to follow in writing interrupt service routines is this: keep your interrupt database separate from your mainline database. If the ISR can freely write to variables used by the mainline, you will certainly have problems when the mainline attempts to work with variables whose values change in unpredictable ways. You must set up ironclad rules about when the ISR can mess with variables used by the mainline, what it can do to them, and how it notifies the mainline routines that it has indeed altered them.

Approach interrupts with extreme caution. They are very powerful, but every programmer can tell you horror stories about debugging interrupt routines.

==========

DAVE'S DISK (Continued from page 3)

At 4:45 there will be a special meeting of the ATR8000-CP/M SIG. We will discuss the status and future plans of this group.

CALLING ALL OFFICER MATERIAL

I will be selecting a nomination committee at the February board meeting. I would like anyone interested on being on the nominating committee, or better yet, anyone who would like to be a candidate to see me at the meeting. The elected positions are President, Vice President, Secretary and Treasurer. The report of the nominating committee will be presented at the April meeting and elections will be help in May.

CALLING ALL BUDDING AUTHORS

You write programs, you know they're good, but you don't know where to go from there.

A communique received from DATASOFT asks for submissions. They want to see original graphic adventure games, action adventure games and action arcade style games full of color and animation. Also, home productivity programs with mass market appeal. Prefer programs written in machine language, but will consider outstanding submissions in BASIC. Programs should be written for ATARI, APPLE][, COMMODORE 64, IBM PC, TANDY 1000 OR TANDY COLOR COMPUTER. They also want products for ATARI ST, MACINTOSH and amiga. Graphics is very important.

Send your submissions to;

DATASOFT
Marketing Project Manager
19808 Nordhoff Place
Chatsworth, CA 91311
(818) 886-5922

DATASOFT promises you a prompt reply.

Worldwide™ Users Network

** Professional GEM **
by Tim Oren
Topic: WINDOWS, part I

ANTIC is proud to present the first of Tim Oren's bi-monthly columns exploring the GEM programming environment. These columns are aimed at professional ST developers, but we encourage everyone to join in and collect the columns for future reference.

HELLO, WORLD!

For those whom I have not met in person or electronically, an introduction is in order.

I am a former member of the GEM programming team at Digital Research, Inc., where I designed and implemented the GEM Resource Construction Set and other parts of the GEM Programmer's Toolkit. I have since left DRI to become the user interface d .gner for Activenture, a startup company which is developing CD-ROM technology for use with the Atari ST and other systems.

The purpose of Professional GEM is to pass along some of the information and tricks I have accumulated about GEM, and explore some of the user interface techniques which a powerful graphics processor such as the ST makes possible.

GROUND RULES

I am going to assume that you have both a working knowledge of the C programming language and a copy of the ST Programmer's Toolkit with documentation (available from Atari). If you lack either, don't panic. You can read the columns to get the flavor of programming the ST, and come back for a more serious visit later on.

For now, I will be using code samples that will run with the Atari-supplied C compiler, also known as DR C-68K, or Alcyon C. I will be using the portability macros supplied with the Toolkit, so that the code will also be transferable to other GEM s .tems.

Both of these items are subject to change, depending on reader feedback and the availability of better products.

If you do not have a copy of the source to the DOODLE.C GEM example program, you should consider downloading a copy from SIG*ATARI (or get a copy from the MILATARI SIG library -ed). Although it is poorly documented, it shows real-life examples of many of the techniques I will discuss.

Getting started with a windowed graphics system seems to be like getting into an ice-cold swimming pool: it's best done all at once.

Anyone who has looked at "Inside Macintosh" has probably noticed that you have to have read most of it to understand any of it. GEM isn't really much different. You have all the reference guides in your hand, but nothing to show how it all works together.

I am hoping to help this situation by leading a series of short tours through the GEM jungle. Each time we'll go out with a particular goal in mind and follow the path that leads there. We'll look at the pitfalls and strange bugs that lurk for the unwary, and show off a few tricks to amaze the natives. The first trip leaves immediately; our mission is to get a window onto the ST screen, with all of its parts properly initialized.

WE DO WINDOWS

One of the most important services which a graphics interface system provides for the user and programmer is window management.

Windows allow the user to perform more than one activity on the same screen, to freely reallocate areas of the screen for each task, and even to pile the information up like pages of paper to make more room. The price for this increased freedom is (as usual) paid by you, the programmer, who must master a more complex method of interacting with the "outside world".

The windowing routines provided by ST GEM are the most comprehensive yet available in a low-cost microcomputer. This article is a guide to using these services in an effective manner.

IN THE BEGINNING

In GEM, creating a window and displaying it are two different functions. The creation function is called wind_create, and its calling sequence is:

```
handle = wind_create(parts, xfull, yfull,
wfull, hfull);
```

This function asks GEM to reserve space in its memory for a new window description, and to return a code or "handle" which you can use to refer to the window in the future. Valid window handles are positive integers; they are not memory pointers.

GEM can run out of window handles. If it does so, the value returned is negative. Your code should always check for this situation and ask the program's user to close some windows and retry if possible. Handle zero is special. It refers to the "desktop", which is predefined as light green (or gray) on the ST. Window zero is always present and may be used, but never deleted, by the programmer.

The xfull, yfull, wfull, and hfull parameters are integers which determine the maximum size of the window. Xfull and yfull define the upper left corner of the window, and wfull and hfull specify its width and height. (Note that all of the window coordinates which we use are in pixel units.)

GEM saves these values so that the program can get them later when processing FULL requests. Usually the best maximum size for a window is the entire desktop area, excepting the menu bar. You can find this by asking wind_get for the working area of the desktop (handle zero, remember):

```
wind_get(0, WF_WXYWH, &xfull, &yfull, &wfull, &hfull);
```

Note that WF_WXYWH, and all of the other mnemonics used in this article, are defined in the GEMDEFS.H file in the ST Toolkit.

The parts parameter of wind_create defines what features will be included in the window when it is drawn. It is a word of single bit flags which indicate the presence/absence of each feature. To request multiple features, the flags are "or-ed" together. The flags' mnemonics and meanings are:

NAME- A one character high title bar at the top of the window.

INFO- A second character line below the NAME.

MOVER- This lets the user move the window around by "dragging" in the NAME area. NAME also needs to be defined.

CLOSER - A square box at the upper left. Clicking this control point asks that the window be removed from the screen.

FULLER - A diamond at upper right. Clicking this control point requests that th window grow to its maximum size, or shrink back down if it is already big.

SIZER - An arrow at bottom right. Dragging the SIZER lets the user choose a new size for the window.

VSLIDE - defines a right-hand scroll box and bar for the window. By draggingthe scroll bar, the user requests that the window's "viewport" into the information be moved. Clicking on the gray box above the bar requests that the window be moved up one "page". Clicking below the bar requests a down page movement. You have to define what constitutes a page or line in the context of your application.

UPARROW - An arrow above the right scroll bar. Clicking here requests that the window be moved up one "line". Sliders and arrows almost always appear together.

DNARROW - An arrow below the right scroll bar. Requests that window be moved down a line.

HSLIDE - These features are the horizontal equivalent of the RTARROW above. They appear at the bottom of the window. Arrows LFARR(usually indicate "character" sized movement left and right. "Page" sized movement has to be defined by each application.

It is important to understand the correspondence between window features and event messages which are sent to the application by the GEM window manager. If a feature is not included in a window's creation, the user cannot perform the corresponding action, and your application will never receive the matching message type. For example, a window without a MOVER may not be dragged by the user, and your app will never get a WM_MOVED message for that window.

Another important principle is that the application itself is responsible for implementing the user's window action request when a message is received. This gives the application a chance to accept, modify, or reject the user's request.

As an example, if a WM_MOVED message is received, it indicates that the user has dragged the window. You might want to byte or word align the requested position befor proceeding to move the window. The wind_se calls used to perform the actual movements will be described in the next article.

OPEN, SESAME!

The wind_open call is used to actually make the window appear on the screen. It animates a "zoom box" on the screen and then draws in the window's frame. The calling sequence is:

```
wind_open(handle, x, y, w, h);
```

handle is the one returned by wind_create. Parameters x, y, w, and h define the initial location and size of the window. Note that these measurements INCLUDE all of the window frame parts which you have requested. To find out the size of the area inside the frame, you can use

```
wind_get(handle, WF_UXYUH, &inner_x, &inner_y, &inner_w, &inner_h);
```

Whatever size you choose for the window display, it cannot be any larger than the full size declared in wind_create.

Here is a good place to take note of a useful utility for calculating window sizes. If you know the "parts list" for a window, and its inner or outer size, you can find the other size with the wind_calc call:

```
wind_calc(parts, kind, input_x, input_y, input_w, input_h, &output_x, &output_y, &output_w, &output_h);
```

Kind is set to zero if the input coordinates are the inner area, and you are calculating the outer size. Kind is one if the inputs are the outer size and you want the equivalent inner size. Parts are just the same as in wind_create.

There is one common bug in using wind_open. If the NAME feature is specified, then the window title must be initialized BEFORE opening the window:

```
wind_set(handle, WF_NAME, ADDR(title), 0, 0);
```

If you don't do this, you may get gibberish in the NAME area or the system may crash. Likewise, if you have specified the INFO feature, you must make a wind_set call for WF_INFO before opening the window.

Note that ADDR() specifies the 32-bit address of title. This expression is portable to other (Intel-based) GEM systems. If you don't care about portability, then &title[0], or just title alone will work fine on the ST.

CLEANING UP.

When you are done with a window, it should be closed and deleted. The call wind_close(handle); takes the window off the screen, redraws the desktop underneath it, and animates a "zoom down" box. It doesn't delete the window's definition, so you can reopen it later.

Deleting the window removes its definition from the system, and makes that handle available for reuse. Always close windows before deleting, or you may leave a "dead" picture on the screen. Also be sure to delete all of your windows before ending the program, or your app may "eat" window handles. The syntax for deleting a window is:

```
wind_delete(handle);
```

THOSE FAT SLIDERS.

One of ST GEM's unique features is the proportional slider bar. Unlike other windowing systems, this type of bar gives visual feedback on the fraction of a document which is being viewed, as well as the position within the document. The catch, of course, is that you have two variables to maintain for each scroll bar: size and position.

Both bar size and position range from 1 to 1000. A bar size of 1000 fills the slide box, and a value of one gets the minimum bar size. To compute the proper size, you can use the formula:

```
size = min(1000, 1000 * seen_doc / total_doc)
```

Seen_doc and total_doc are the visible and total size of the document respectively, in whatever units are appropriate. As an example, if your window could show 20 lines of a 100 line text file, you should set a slider size of 200. Since the window might be bigger than the total document at some points, you need the maximum function. If the document size is zero, force the slider size to 1000. (Note: You will probably need to do the computation above with 32-bit arithmetic to avoid overflow problems.)

Once you have computed the size, use the wind_set function to configure the scroll bar:

```
wind_set(handle, WF_VSLSIZE, size, 0, 0, 0);
```

This call sets the vertical (right hand) scroll bar. Use WF_HSLSIZE for the horizontal scroller. All of these examples are done for the vertical dimension, but the principles are identical in the other direction.

Bar positioning is a little tougher. The most confusing aspect is that the 1-1000 range does not set an absolute position of the bar within the scroll box. Instead, it positions the TOP of the bar within its possible range of variation.

Let's look at our text file example again to make this clearer. If there arealways 20 lines of a 100 line file visible, then the top of the window must be always be somewhere between line 1 and line 81. This 80 line range is the actual freedom of movement of the window. So, if the window were actually positioned with its top at line 61, it would be at the three-quarter position within the range, and we should set a scroll bar position of 750. The actual formula for computing the position is:

```
pos = 1000 * (top_wind - top_doc) / (total_doc - seen_doc)
```

Top_wind and top_doc are the top line in the current window and the whole document, respectively. Obviously, if seen_doc is greater or equal to total_doc, you need to force a zero value for pos. This calculation may seem rather convoluted the first time through, but is easy once you have done it. When you have computed the position, wind_set configures the scroll bar:

```
wind_set(handle, WF_VSLIDE, pos, 0, 0, 0);
```

WF_HSLIDE is the equivalent for horizontal scrolling.

It is a good practice to avoid setting the slider size or position if they are already at the value which you need. This avoids an annoying redraw flash on the screen when it is not necessary. You can check on the current value of a slider parameter with wind_get:

```
wind_get(handle, WF_VSLIDE, &curr_value, &foo, &foo, &foo);
```

Foo is a dummy variable which needs to be there, but is not used. Substitute WF_VSLIDE with whatever parameter you are checking.

One philosophical note on the use of sliders: It is probably best to avoid the use of both sliders at once unless it is clearly appropriate to the type of data which is being viewed.

Since Write and Paint programs make use of the sheet-of-paper metaphor, moving the window around in both dimensions is reasonable. However, if the data is more randomly organized, such as a tableau of icons, then it is probably better to only scroll in the vertical dimension and "reshuffle" if the window'swidth is changed. Then the user only needs to manipulate one control to find information which is off-screen. Anyone who has had trouble finding a file or folder within a Desktop window will recognize this problem.

COMING UP NEXT

In my next column in Antic Online, we'll conclude the tour of the ST's windowing system. I'll discuss the correct way to redraw a window's contents, and how to handle the various messages which an application receives from the window manager. Finally, we'll look at a way to redesign the desktop background to your own specifications.

Reprinted from A.C.T. Toledo

The Man From A.S.C.I.I.
REPORT #0985
E.Z. HINTZ REPORTING...

SEND ALL QUESTIONS TO:
MAN FROM A.S.C.I.I.
C/O ROBERT WROBEL
606 CARLTON
TOLEDO, OHIO 43609

ADVENTURE: THE INSTITUTE

QUESTION: I KNOW THAT I NEED A PLANT TO STAY IN THE STATUE'S HEAD. MY QUESTION IS WERE IS THE PLANT, I CANNOT FIND IT ?

HINT: FOR THIS ONE I HAD TO GO TO A HIGHER SOURCE- THE HINT LORD. WE COMBINED OUR WISDOM AND KNOWLEDGE TO COME UP WITH A one word HINT:
T H I R S T Y

ADVENTURE: THE RETURN OF HERACLES

QUESTION: IS HERACLES TO BE FOUND?

HINT: THE STRONGEST MAN WILL OBEY YOUR COMMANDS IF YOU START WITH THE CORRECT CHARACTER AND VIST THE RIGHT ORACLE. STUDY YOUR MYTHOLOGY!

MILWAUKEE AREA ATARI USER'S GROUP AND NEWSLETTER INFORMATION

MILATARI OFFICERS

President	David Frazer	542-7242
Vice President	Carl Mielcarek	355-3539
Secretary	Steve Armstrong	543-9039
Treasurer	Steve Tupper	462-8178

MILATARI VOLUNTEERS

MILATARI West	Dennis Bogie	968-2361
Education SIG	Joe Sanders	447-1660
ATR8000 & CP/M	Joe Kasper	782-9041
Adv. Lang. SIG	Erik Hanson	252-3146
Database	Ron Friedel	354-1717
BBS Sysop	Richard Dankert	781-2338
Membership	Dave Bogie	968-2361
Newsletter	Roy Duvall	363-8231
Public Domain Libraries		
Cassette	Lee Musial	835-1144
Disk	Dennis Wilson	476-7767
	Bill Lawrence	968-9341
Copyright Library		
Cassette/Disk	Gary Haberman	228-8845
Publications	Bill Feest	321-4314

Milatari BBS

300 Baud 24hrs. 414-781-5710

NEWSLETTER INFORMATION

This newsletter is written and printed by members of the Milwaukee Area ATARI User's Group (MILATARI). Opinions expressed in this publication are those of the individual authors and do not necessarily represent or reflect the opinions of the Milwaukee Area Atari User's Group, its officers, its members or its advertisers except where noted. Articles reprinted from bulletin boards and other newsletters are presented as an information exchange, no warranty is made to their accuracy or factuality.

Your contributions of articles are always welcome. You may submit your article on ATARI compatible cassette or diskette, on typewritten form or you can arrange with the editor to upload your file via modem. You can send Graphics eight or seven plus screens stored on disk in Micropainter or Micro Illustrator formats.

Milwaukee Area Atari User's Group

MILATARI is an independent, user education group which is not affiliated with ATARI INC. The newsletter is the official publication of MILATARI and is intended for the education of its members as well as for the dissemination of information concerning ATARI computer products.

MILATARI membership is open to individuals and families who are interested in using and programming ATARI computers. The membership includes a subscription to this newsletter and access to the club libraries. The annual membership fee is $15 for individuals or $20 for a family.

Vendors wishing to display and/or sell items at MILATARI meetings must make prior arrangements with the club vice president. Rates are $10 per meeting or $90 per year payable in advance.

Other computer user groups may obtain copies of this newsletter on an exchange basis.

MILATARI ADVERTISING RATES

This newsletter will accept camera ready advertising copy from anyone supplying goods and services of interest to our membership.

Current paid members of MILATARI may place classified ads in the newsletter at no charge.

Advertising Rates

Full page	$37.50
Half page	$20.00
Quarter page	$12.50
Business card	$2.00

MILWAUKEE AREA ATARI USERS CLUB

U.S. POSTAGE PAID
BULK RATE
PERMIT NO. 201
WAUKESHA, WI 53187

MILWAUKEE AREA ATARI USERS GROUP
Post Office Box 19858
West Allis, Wisconsin 53219-0858

Address Correction Requested
Forwarding Postage Guaranteed